



Langage OCL (Object Constraint Language)

Ouassila Labbani-Narsis

ouassila.narsis@u-bourgogne.fr

Pourquoi OCL ?

Introduction par l'exemple

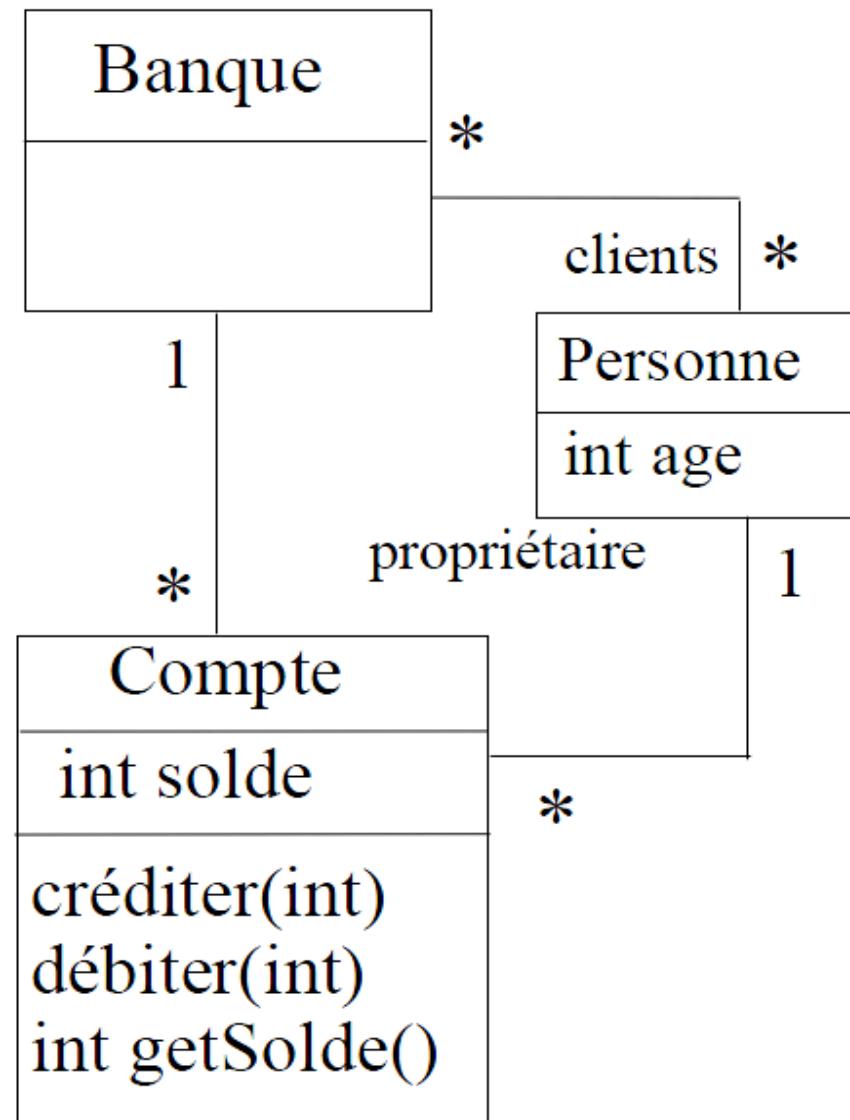
Application bancaire

- Des comptes bancaires
- Des clients
- Des banques

Spécification informelle

- Un compte doit avoir un solde toujours positif
- Un client peut posséder plusieurs comptes
- Un client peut être client de plusieurs banques
- Un client d'une banque possède au moins un compte dans cette banque
- Une banque gère plusieurs comptes
- Une banque possède plusieurs clients

Diagramme de classe



Manque de précision ...

- Le diagramme de classe ne permet pas d'exprimer tout ce qui est défini dans la spécification informelle
- Exemple
 - Le solde d'un compte doit toujours être positif
 - ➔ ajout d'une contrainte sur cet attribut
 - Le diagramme de classe permet-il de détailler toutes les contraintes sur les relations entre les classes ?

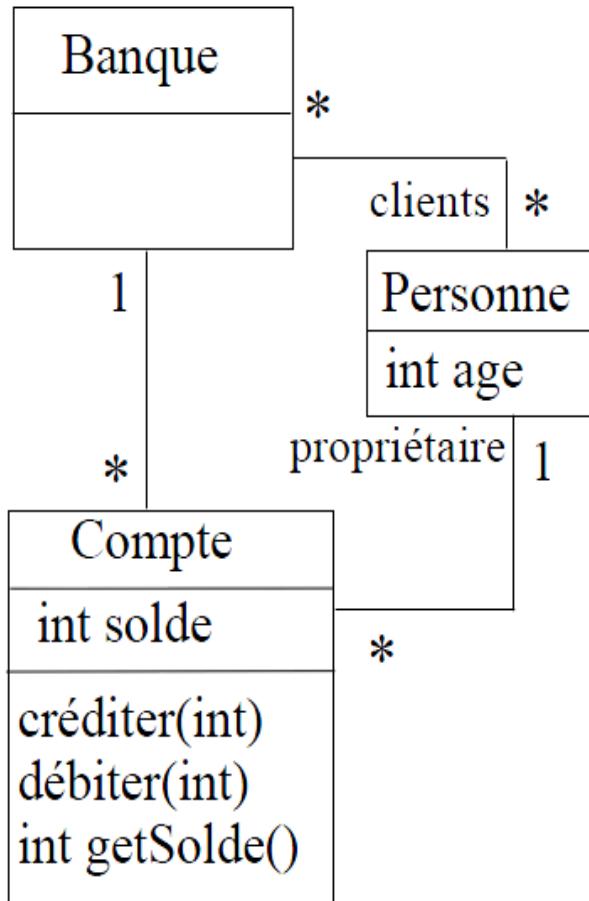
Diagrammes UML insuffisants

- Pour spécifier complètement une application
 - Diagrammes UML seuls sont généralement insuffisants
 - Nécessité de rajouter des contraintes
- Comment exprimer ces contraintes ?
 - Langue naturelle mais manque de précision
 - Compréhension pouvant être ambiguë
 - Langage formel avec sémantique précise
 - Exemple : OCL
- **OCL** : Object Constraint Language
 - Langage de contraintes orienté-objet
 - Langage formel (mais simple à utiliser) avec une syntaxe, une grammaire et une sémantique
 - Manipulable par des outils
 - S'applique entre autres sur les diagrammes UML

Le langage OCL

- Peut s'appliquer sur tout type de modèle, indépendant d'un langage de modélisation donné
- OCL permet principalement d'exprimer deux types de contraintes sur l'état d'un objet ou d'un ensemble d'objets
 - Des **invariants** qui doivent être respectés en permanence
 - Des **pré et post-conditions** pour une opération
 - **Pré-condition** : doit être vérifiée avant l'exécution
 - **Post-condition** : doit être vérifiée après l'exécution
- **Attention** : Une expression OCL décrit une contrainte à respecter et non pas le « code » d'une méthode

Usage d'OCL sur l'application bancaire



context Compte

inv: solde > 0

context Compte :: débiter(somme : int)

pre: somme > 0

post: solde = solde@pre - somme

context Compte

inv: banque.clients

-> includes (propriétaire)

Avantage d'OCL : langage formel permettant de préciser clairement la sémantique sur les modèles UML

Utilisation d'OCL dans le cadre d'UML

- OCL peut s'appliquer sur la plupart des diagrammes UML
- Il sert, entre autres, à spécifier des
 - Invariants sur des classes
 - Pré et post-conditions sur des opérations
 - Gardes sur transitions de diagrammes d'états ou de messages de diagrammes de séquence/collaboration
 - Des ensembles d'objets destinataires pour un envoi de message
 - Des attributs dérivés
 - Des stéréotypes
 - ...

Contexte

- Une expression OCL est toujours définie dans un **contexte**
- Mot-clé : **context**
- **Exemple**
 - **context** Compte
 - L'expression OCL s'applique à la classe Compte, c'est-à-dire à toutes les instances de cette classe

Invariants

- Un **invariant** exprime une contrainte sur un objet ou un groupe d'objets qui doit être respectée en permanence
- Mot-clé : **inv**
- Exemple
 - **context** Compte
inv: $\text{solde} > 0$
 - Pour toutes les instances de la classe Compte, l'attribut solde doit toujours être positif

Pré et post-conditions

- Pour spécifier une opération :
 - **Pré-condition** : état qui doit être respecté avant l'appel de l'opération
 - **Post-condition** : état qui doit être respecté après l'appel de l'opération
- **Mots-clés** : **pre** et **post**
- Dans la post-condition, deux éléments particuliers sont utilisables :
 - L'attribut **result** : référence la valeur renvoyée par l'opération
 - *mon_attribut* **@pre** : référence la valeur de *mon_attribut* avant l'appel de l'opération
- **Syntaxe pour préciser l'opération**
 - **context** ma_classe::mon_op(liste_param) : type_retour 11

Pré et post-conditions

Exemples

- **context** Compte::débiter(somme : int)
pre: somme > 0
post: solde = solde@**pre** – somme
 - La somme à débiter doit être positive pour que l'appel de l'opération soit valide
 - Après l'exécution de l'opération, l'attribut **solde** doit avoir pour valeur la différence de sa valeur avant l'appel et de la somme passée en paramètre
- **context** Compte::getSolde () : int
post: **result** = solde
 - Le résultat retourné doit être le solde courant

Attention : On ne décrit pas comment l'opération est réalisée mais des contraintes sur l'état avant et après son exécution

Accès aux objets, navigation

- Dans une contrainte OCL associée à un objet, on peut :
 - Accéder à l'état interne de cet objet (ses attributs)
 - Naviguer dans le diagramme : accéder de manière transitive à tous les objets (et leur état) avec qui il est en relation

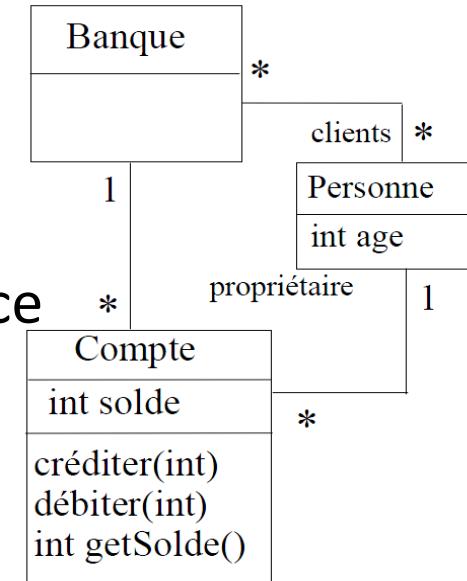
Nommage des éléments

- Attributs ou paramètres d'une opération : utilise leur nom directement
- Objet(s) en association : utilise le nom de la classe associée (en minuscule) ou le nom du rôle d'association du côté de cette classe
- Si la cardinalité est de **1** pour une association : référence un seul objet
- Si la cardinalité **> 1** : référence une collection d'objets

Accès aux objets, navigation

Exemples : dans le contexte de la classe Compte :

- solde** : attribut référencé directement
- banque** : objet de la classe Banque (référence via le nom de la classe) associé au compte
- propriétaire** : objet de la classe Personne (référence via le nom de rôle d'association) associée au compte
- banque.clients** : ensemble des clients de la banque associée au compte (référence par transitivité)
- banque.clients.age** : ensemble des âges de tous les clients de la banque associée au compte



Exemple : Le propriétaire d'un compte doit avoir plus de 18 ans

- context** Compte
inv: propriétaire.age >= 18

Opérations sur objets et collections

- OCL propose un ensemble de primitives utilisables sur les collections
 - **size()** : retourne le nombre d'éléments de la collection
 - **isEmpty()** : retourne vrai si la collection est vide
 - **notEmpty()** : retourne vrai si la collection n'est pas vide
 - **includes(obj)** : vrai si la collection inclut l'objet *obj*
 - **excludes(obj)** : vrai si la collection n'inclut pas l'objet *obj*
 - **including(obj)** : la collection référencée doit être cette collection en incluant l'objet *obj*
 - **excluding(obj)** : idem mais en excluant l'objet *obj*
 - **includesAll(ens)** : la collection contient tous les éléments de la collection *ens*
 - **excludesAll(ens)** : la collection ne contient aucun des éléments de la collection *ens*
- Syntaxe d'utilisation : ***objetOuCollection -> primitive*** 15

Opérations sur objets et collections

Exemples: invariants dans le contexte de la classe Compte

- **propriétaire** -> **notEmpty()** : il y a au moins un objet Personne associé à un compte
 - **propriétaire** -> **size() = 1** : le nombre d'objets Personne associés à un compte est de 1
 - **banque.clients** -> **size() >= 1** : une banque a au moins un client
 - **banque.clients** -> **includes(propriétaire)** : l'ensemble des clients de la banque associée au compte contient le propriétaire du compte
 - **banque.clients.compte** -> **includes (self)** : le compte appartient à un des clients de sa banque
- **self**
- pseudo-attribut référençant l'objet courant

Opérations sur objets et collections

Un autre exemple:

```
context Banque :: creerCompte(p : Personne) : Compte
post: result.oclIsNew() and
compte = compte@pre -> including(result) and
p.compte = p.compte@pre -> including(result)
```

- Un nouveau compte est créé. La banque doit gérer ce nouveau compte. Le client passé en paramètre doit posséder ce compte. Le nouveau compte est retourné par l'opération
- **OclIsNew()**
 - Primitive indiquant qu'un objet doit être créé pendant l'appel de l'opération (à utiliser dans une post-condition)
- **and** : Permet de définir plusieurs contraintes pour un invariant, une pré ou post-condition
 - and = « et logique » : l'invariant, pré ou post-condition est vrai si toutes les expressions reliées par le « and » sont vraies

Relations ensemblistes entre collections

Deux relations : **union** et **intersection**

- **Union** : Retourne l'union de deux collections
- **Intersection** : Retourne l'intersection de deux collections

Exemples

- `(col1 -> intersection(col2)) -> isEmpty ()`
 - Renvoie vrai si les collections col1 et col2 n'ont pas d'élément en commun
- `col1 = col2 -> union (col3)`
 - La collection *col1* doit être l'union des éléments de *col2* et de *col3*

Opérations sur éléments d'une collection

- OCL permet de vérifier des contraintes sur chaque élément d'une collection ou de définir une sous-collection à partir d'une collection en fonction de certaines contraintes
- Exemples de primitives s'appliquant sur une collection *col* :
 - **select** : retourne le sous-ensemble de la collection *col* dont les éléments respectent la contrainte spécifiée
 - **reject** : idem mais ne garde que les éléments ne respectant pas la contrainte
 - **collect** : retourne une collection (de taille identique) construite à partir des éléments de *col*. Le type des éléments contenus dans la nouvelle collection peut être différent de celui des éléments de *col*
 - **exists** : retourne vrai si au moins un élément de *col* respecte la contrainte spécifiée et faux sinon
 - **forAll** : retourne vrai si tous les éléments de *col* respectent la contrainte spécifiée (pouvant impliquer à la fois plusieurs éléments de la collection)

Opérations sur éléments d'une collection

Syntaxe de ces opérations : 3 usages

collection -> primitive(expression)

- La primitive s'applique aux éléments de la collection et pour chacun d'entre eux, l'expression *expression* est vérifiée. On accède aux attributs/relations d'un élément directement

collection -> primitive(elt:type|expression)

- On fait explicitement apparaître le type des éléments de la collection (ici *type*). On accède aux attributs/relations de l'élément courant en utilisant *elt* (c'est la référence sur l'élément courant)

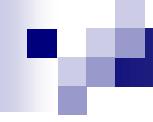
collection -> primitive(elt|expression)

- On nomme l'attribut courant (*elt*) mais sans préciser son type

Opérations sur éléments d'une collection

Dans le contexte de la classe **Banque** :

- compte -> **select(c | c.solde > 1000)**
 - Retourne une collection contenant tous les comptes bancaires dont le solde est supérieur à 1000 €
- compte -> **reject (solde > 1000)**
 - Retourne une collection contenant tous les comptes bancaires dont le solde n'est pas supérieur à 1000 €
- compte -> **collect (c : Compte | c.solde)**
 - Retourne une collection contenant l'ensemble des soldes de tous les comptes
- (compte -> **select (solde > 1000)**)
-> **collect (c | c.solde)**
 - Retourne une collection contenant tous les soldes des comptes dont le solde est supérieur à 1000 €



Opérations sur éléments d'une collection

□ context Banque

inv: **not**(clients -> **exists** (age < 18))

- Il n'existe pas de clients de la banque dont l'âge est inférieur à 18 ans
- **not** : prend la négation d'une expression

□ context Personne

inv : Personne.**allInstances** () -> **forall** (p1, p2 | p1 <> p2 **implies** p1.nom <> p2.nom)

- Il n'existe pas deux instances de la classe Personne pour lesquelles l'attribut nom a la même valeur : deux personnes différentes ont un nom différent
- **AllInstances ()** : primitive s'appliquant sur une classe (et non pas un objet) et retournant toutes les instances de la classe référencée (ici la classe Personne)

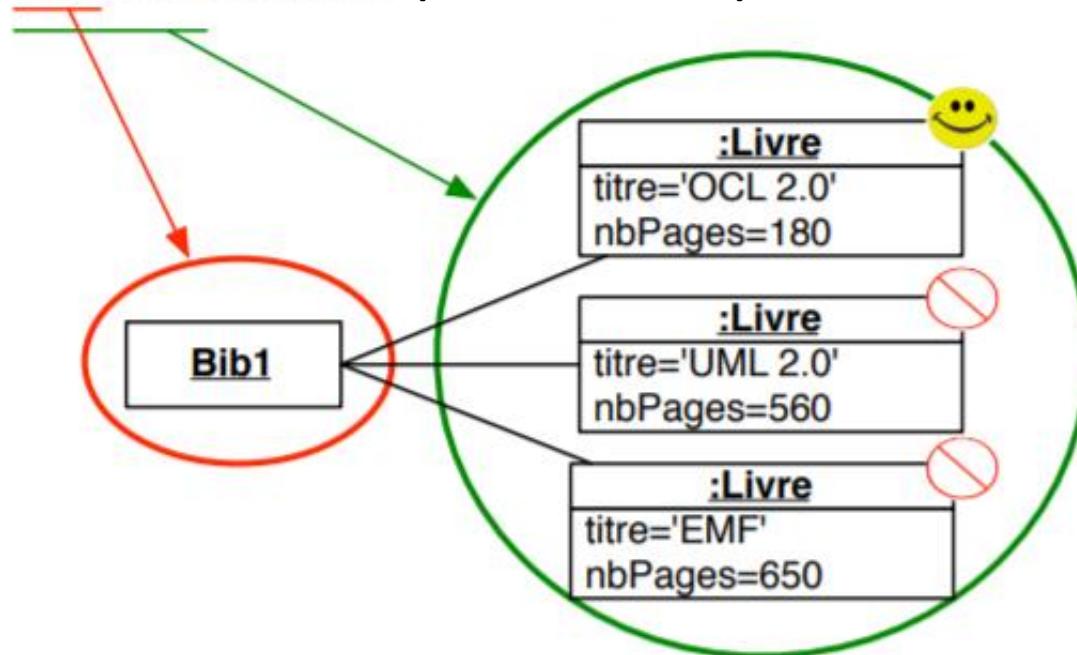
Opérations sur éléments d'une collection

Exemple d'une bibliothèque



context Bibliothèque inv:

```
self.livres->exists( titre = 'OCL 2.0' )
```



La contrainte est vérifiée

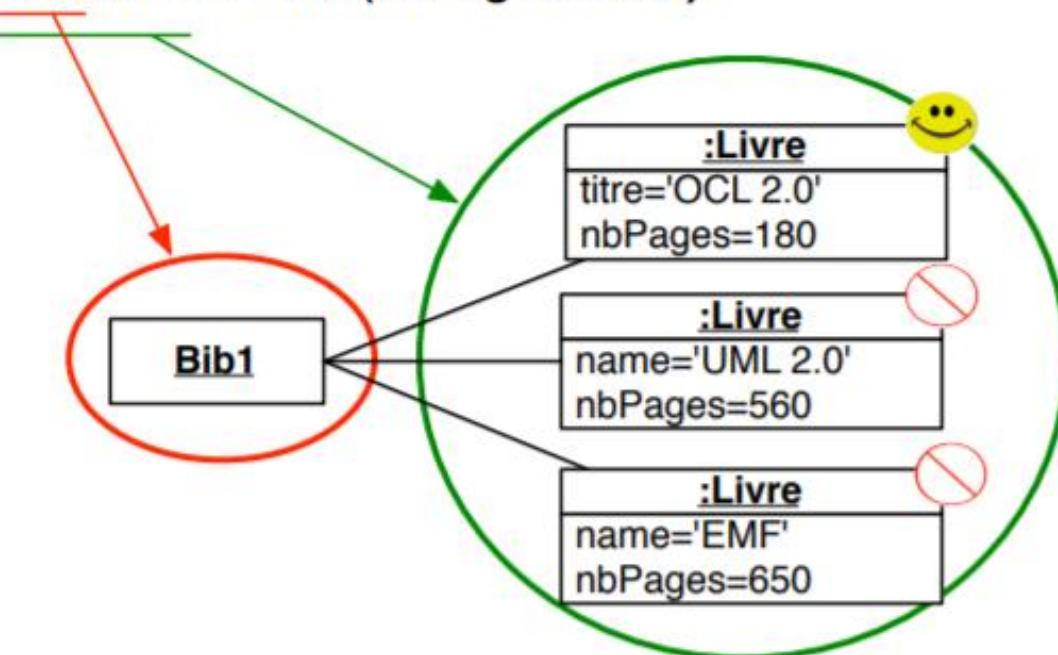


Opérations sur éléments d'une collection

Exemple d'une bibliothèque



context Bibliothèque inv:
self.livres->forAll(nbPages < 200)



La contrainte n'est pas vérifiée

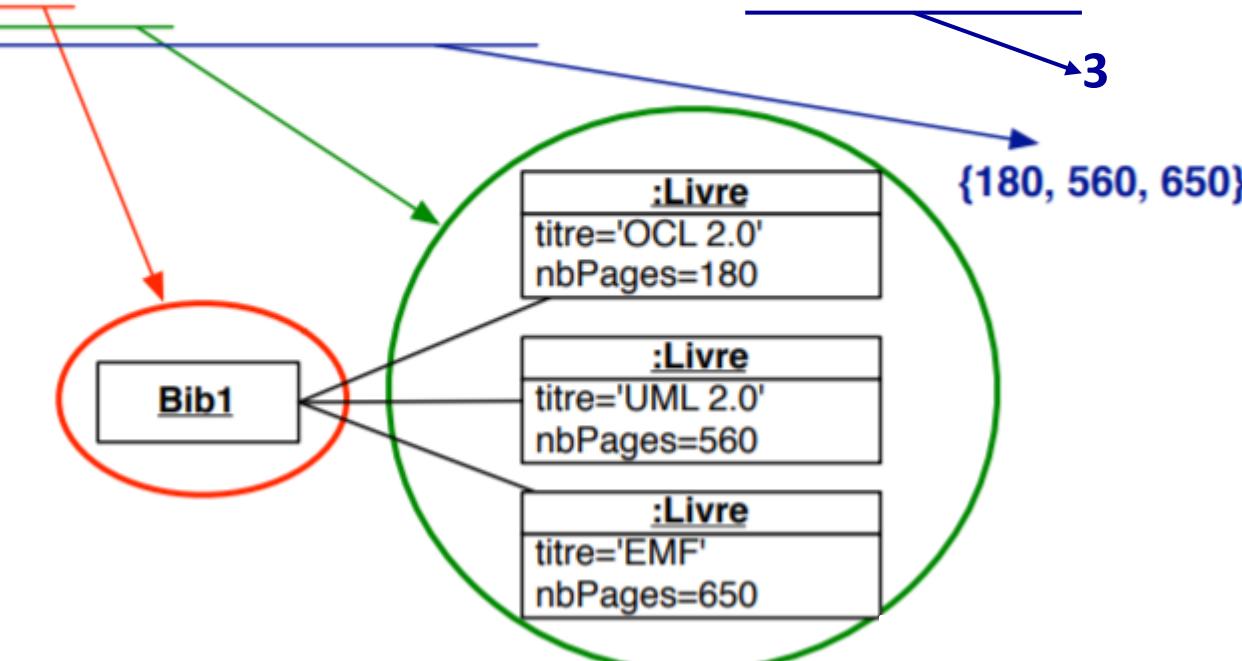


Opérations sur éléments d'une collection

Exemple d'une bibliothèque



```
context Bibliothèque def moyenneDesPages : Real =  
    self.livres->collect(nbPages)->sum() / self.livres->size() > 400
```



La contrainte est vérifiée



Types OCL : types de base

Types de base et exemples d'opérations associées

□ Integer

- 1, -2, 145
- *, +, -, /, abs()

□ Real

- 1.5, -123.4
- *, +, -, /, floor()

□ String

- 'bonjour'
- concat(), size(), substring()

□ Boolean

- true, false
- And, or, not, xor, not, implies, if-then-else
- La plupart des expressions OCL sont de types Booléen
- Notamment les expressions formant les inv, pre et post

Types OCL : types de collection

□ 4 types de collection d'objets

- **Set** : ensemble au sens mathématique, pas de doublons, pas d'ordre
- **OrderedSet** : idem mais avec ordre
- **Bag** : comme un Set mais avec possibilité de doublons
- **Sequence** : un Bag dont les éléments sont ordonnés

□ Exemples :

- { 1, 4, 3, 5 } : Set
- { 1, 4, 1, 3, 5, 4 } : Bag
- { 1, 1, 3, 4, 4, 5 } : Sequence

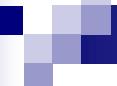
□ Notes

- Un **collect()** renvoie toujours un Bag
- Possibilité de transformer un type de collection en un autre type de collection avec opérations OCL dédiées

Types OCL : types de collection

Collections imbriquées

- Via navigation, on peut récupérer des collections ayant pour éléments d'autres collections
- Deux modes de manipulation
 - Explicitement comme une collection de collections
 - Collection unique : on « aplatit » le contenu de toutes les collections imbriquées en une seule à un seul niveau
 - Opération ***flatten()*** pour aplatisr une collection de collections



Types OCL : conformance de types

Conformance de type

- Prise en compte des spécialisations entre classes du modèle
- Opérations OCL dédiées à la gestion des types
 - **oclIsTypeof** (*type*) : vrai si l'objet est du type *type*
 - **oclIsKindof** (*type*) : vrai si l'objet est du type *type* ou un de ses sous-types
 - **oclAsType** (*type*) : l'objet est « casté » en type *type*

Types internes à OCL

- Conformance entre les types de collection
 - **Collection** est le super-type de **Set**, **Bag** et **Sequence**
 - Conformance entre collection et types des objets contenus
 - **Set** (*T1*) est conforme à **Collection** (*T2*) si *T1* est sous-type de *T2*
 - **Integer** est un sous-type de **Real**

Conditionnelles

- Certaines contraintes sont dépendantes d'autres contraintes
- Deux formes pour gérer cela
 - **if** expr1 **then** expr2 **else** expr3 **endif**
 - Si l'expression *expr1* est vraie alors *expr2* doit être vraie sinon *expr3* doit être vraie
 - expr1 **implies** expr2
 - Si l'expression *expr1* est vraie, alors *expr2* doit être vraie également
 - Si *expr1* est fausse, alors l'expression complète est vraie

Conditionnelles

□ **context** Personne **inv** :

```
if age < 18  
then compte -> isEmpty()  
else compte -> notEmpty()  
endif
```

- Une personne de moins de 18 ans n'a pas de compte bancaire alors qu'une personne de plus de 18 ans possède au moins un compte

□ **context** Personne **inv** :

```
compte -> notEmpty()  
implies banque -> notEmpty()
```

- Si une personne possède au moins un compte bancaire, alors elle est cliente d'au moins une banque

Commentaires et nommage de contraintes

□ Commentaire en OCL : utilisation de --

□ Exemple

```
context Personne inv:  
    if age < 18 -- vérifie age de la personne  
    then compte -> isEmpty ()      -- pas majeur : pas de compte  
    else compte -> notEmpty ()     -- majeur : doit avoir  
                                    -- au moins un compte  
endif
```

□ On peut nommer des contraintes

□ Exemple

```
context Compte  
inv soldePositif: solde > 0  
context Compte::débiter(somme : int)  
pre sommePositive: somme > 0  
post sommeDébitée: solde = solde@pre - somme
```

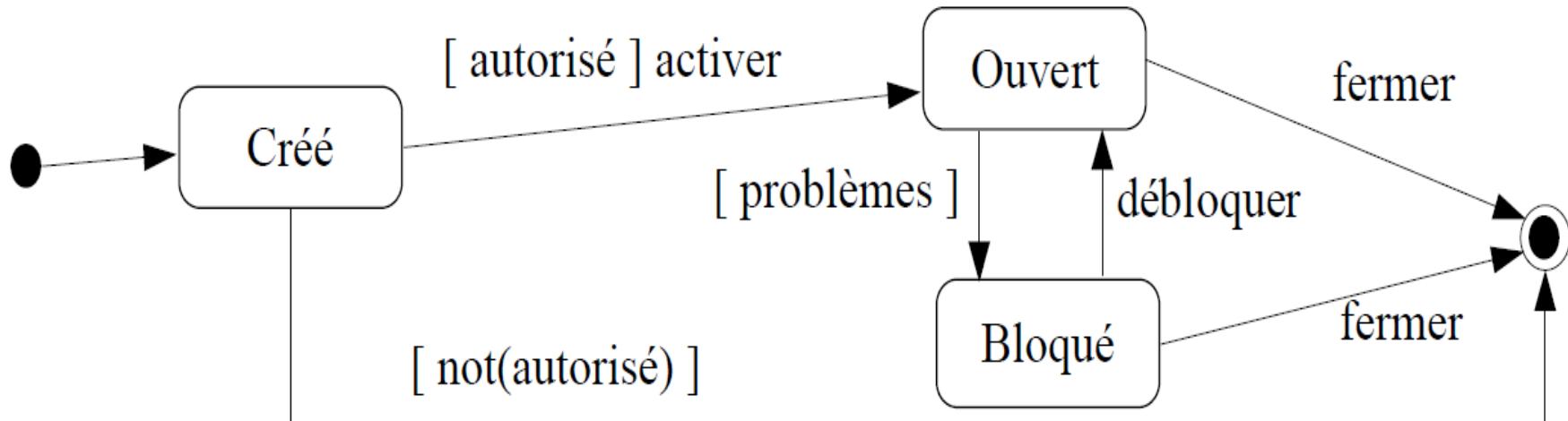
Variables

- Pour faciliter l'utilisation de certains attributs ou calculs de valeurs on peut définir des variables
- Dans une contrainte OCL : **let ... in ...**
 - **context** Personne
 - inv**: let argent = compte.solde -> **sum ()** in
age ≥ 18 **implies** argent > 0
 - Une personne majeure doit avoir de l'argent
 - **sum ()** : fait la somme de tous les objets de la collection
 - Pour l'utiliser partout : **def**
 - **context** Personne
 - def**: argent : int = compte.solde -> **sum ()**
 - **context** Personne
 - inv**: age ≥ 18 **implies** argent > 0

Appels d'opération des classes

- Dans une contrainte OCL : il est possible d'accéder aux attributs, objets ... en lecture
- Possibilité d'utiliser une opération d'une classe dans une contrainte
 - Si pas d'effets de bords
 - Car une contrainte OCL exprime une contrainte sur un état mais ne précise pas qu'une action a été effectuée
- Exemple :
 - **context** Banque
inv: compte -> **forAll(c | c.getSolde() > 0)**
 - **getSolde()** est une opération de la classe Compte. Elle calcule une valeur mais sans modifier l'état d'un compte

Liens avec diagrammes d'états



- Possibilité de référencer un état d'un diagramme d'états associé à l'objet
 - **oclInState (etat)** : vrai si l'objet est dans l'état *etat*
 - Pour sous-états : *etat1::etat2* si *etat2* est un état interne de *etat1*

Exemple

- **context** Compte :: débiter(somme : int)
pre: somme > 0 and **self.oclInState (Ouvert)**
- L'opération débiter ne peut être appelée que si le compte est dans l'état ouvert

Liens avec diagrammes d'états

- On ne peut pas avoir plus de 5 comptes ouverts dans une même banque

context Compte :: activer()

pre: self.oclInState(Créé) and

proprietaire.compte -> **select**(c |

self.banque = c.banque) -> **size()** < 5

post: self.oclInState(Ouvert)

- On peut aussi exprimer la garde [autorisé] en OCL

- **context** Compte

- def:** autorisé : Boolean =

- proprietaire.compte -> **select**(c |

- self.banque = c.banque) -> **size()** < 5

Propriétés

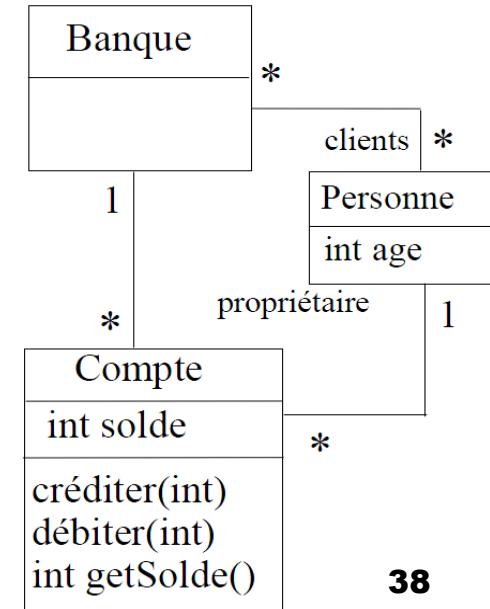
- De manière générale en OCL, une propriété est un élément pouvant être
 - Un attribut
 - Un bout d'association
 - Une opération ou méthode de type requête
- Accède à la propriété d'un objet avec « . »

Exemples

- **context** Compte **inv:** **self.solde > 0**
- **context** Compte **inv:** **self.getSolde() > 0**
- Accède à la propriété d'une collection avec « -> »
 - On peut utiliser « -> » également dans le cas d'un objet (collection d'1 objet)

Accès aux attributs pour les collections

- Accès à un attribut sur une collection
 - Exemple dans contexte de Banque : `compte.solde`
 - Renvoie l'ensemble des soldes de tous les comptes
- Forme raccourcie et simplifiée de
 - `compte -> collect (solde)`



Propriétés prédéfinies en OCL

□ Pour objets :

- **oclIsTypeOf (type)** : l'objet est du type *type*
- **oclIsKindOf (type)** : l'objet est du type *type* ou un de ses sous-types
- **oclInState (état)** : l'objet est dans l'état état
- **oclIsNew ()** : l'objet est créé pendant l'opération
- **oclAsType (type)** : l'objet est « casté » en type *type*

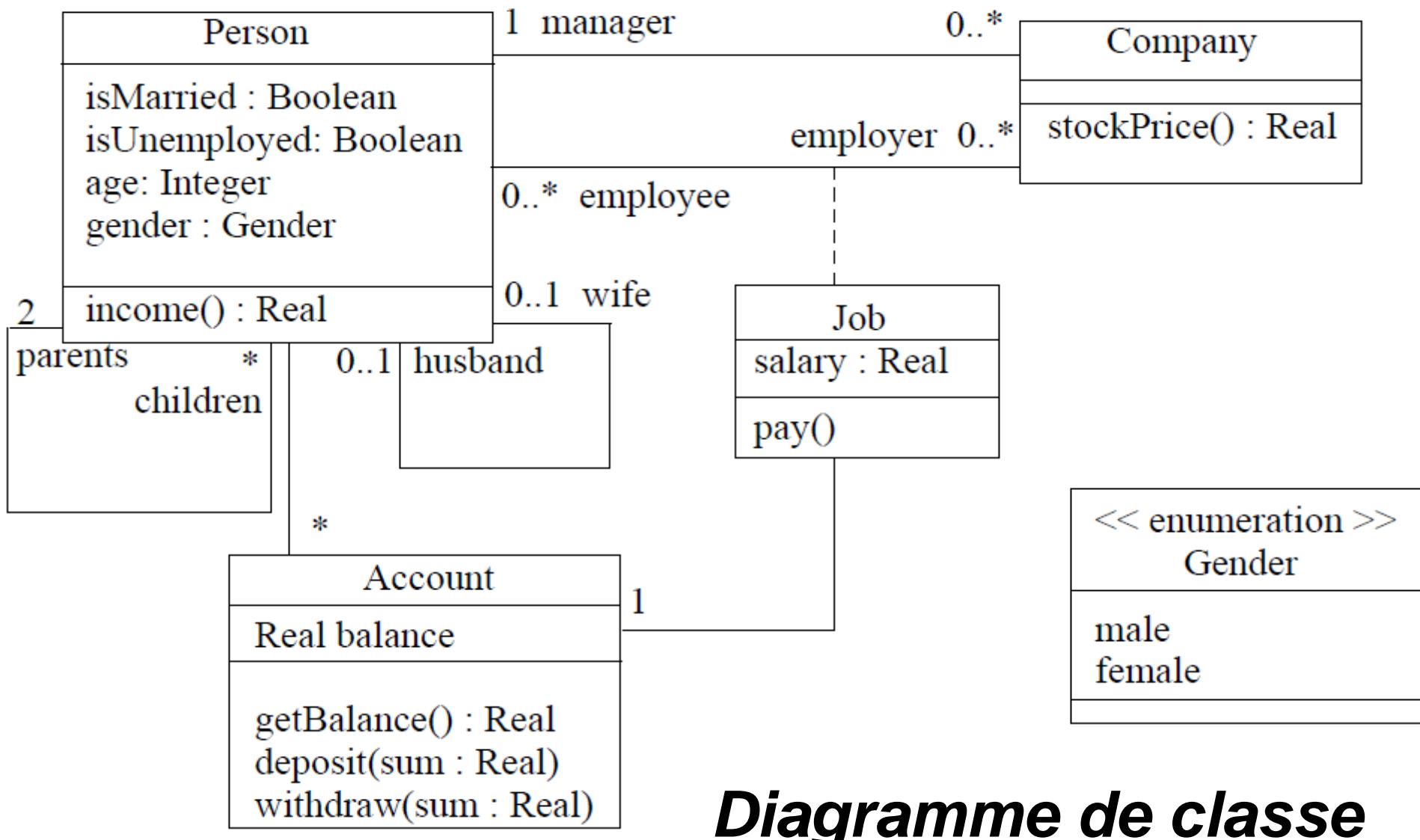
□ Pour collections :

- **isEmpty () , notEmpty () , size () , sum ()**
- **includes () , excludes () , includingAll ()**
-

Règles de précédence

- ❑ Ordre de précédence pour les opérateurs/primitives du plus au moins prioritaire
 - ❑ @pre
 - ❑ . et ->
 - ❑ not et -
 - ❑ * et /
 - ❑ + et -
 - ❑ if then else endif
 - ❑ >, <, <= et >=
 - ❑ = et <>
 - ❑ and, or et xor
 - ❑ implies
- ❑ Parenthèses permettent de changer cet ordre

Exemple d'application



Contraintes sur employés d'une compagnie

- Dans une compagnie, un manager doit travailler et avoir plus de 40 ans
- Le nombre d'employé d'une compagnie est non nul

context Company

inv:

```
self.manager.isUnemployed = false and
self.manager.age > 40 and
self.employee -> notEmpty()
```

Lien salaire/chomage pour une personne

- Une personne considérée comme au chômage ne doit pas avoir des revenus supérieurs à 500 €

```
context Person  
inv:  
let money : Real = self.job.salary->sum() in  
if isUnemployed then  
    money < 500  
else  
    money >= 500  
endif
```

Contraintes sur les parents/enfants

- Un enfant a un père et une mère

context Person

def: parent1 = parents -> at(0)

def: parent2 = parents -> at(1)

context Person

inv:

if parent1.gender = #male

then -- parent1 est un homme

parent2.gender = #female

else -- parent1 est une femme

parent2.gender = #male

endif

Contraintes sur les parents/enfants

- Tous les enfants d'une personne ont bien cette personne comme parent et inversement

context Person

inv:

children -> **notEmpty**()

implies

children -> **forAll** (p : Person | p.parents ->
includes (self))

context Person

inv:

parents -> **forAll** (p : Person | p.children ->
includes (self))

Revenus selon l'âge

- Selon l'âge de la personne, ses revenus sont :
 - 1% des revenus des parents quand elle est mineure (argent de poche)
 - Ses salaires quand elle est majeure

```
context Person::income() : Real
post:
  if age < 18 then
    result = (parents.job.salary -> sum()) / 100
  else
    result = self.job.salary -> sum()
  endif
```

Versement salaire

- Salaire payé :
context Job::pay()
post: account.balance = account.balance@**pre** + salary
- Depuis OCL 2.0 : on peut aussi préciser que l'opération deposit doit être appelée
 - **context** Job::pay()
 - post**: account^deposit(salary)
 - **objet^operation(param1, ...)** : renvoie vrai si un message operation est envoyé à objet avec la liste de paramètres précisée (si pas de valeur particulière : on utilise « ? : type »)

Remarque : On s'éloigne des principes d'OCL (langage de contraintes et pas d'actions). Les actions sont généralement exprimable en UML avec les diagrammes d'interactions (séquence, collaboration)

Outils OCL

Il existe de nombreux outils qui permettent de vérifier la syntaxe, la sémantique et d'évaluer une expression OCL

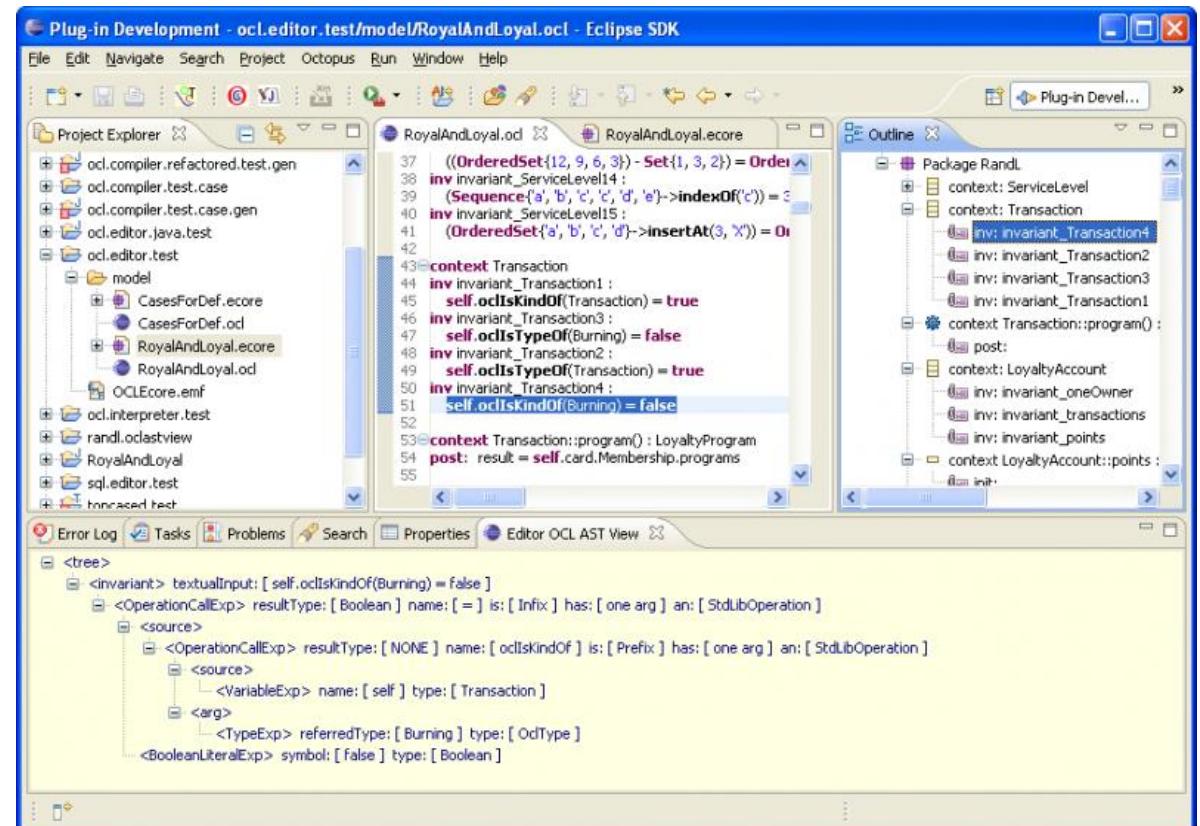


Outils OCL

Quelques outils pour la vérification des contraintes OCL :

❑ Eclipse MDT/OCL (OMG OCL RTF (Revision Task Force))

- CompleteOCL Editor
- OCLinEcore Editor
- EssentialOCL Editor
- OCLstdlib Editor

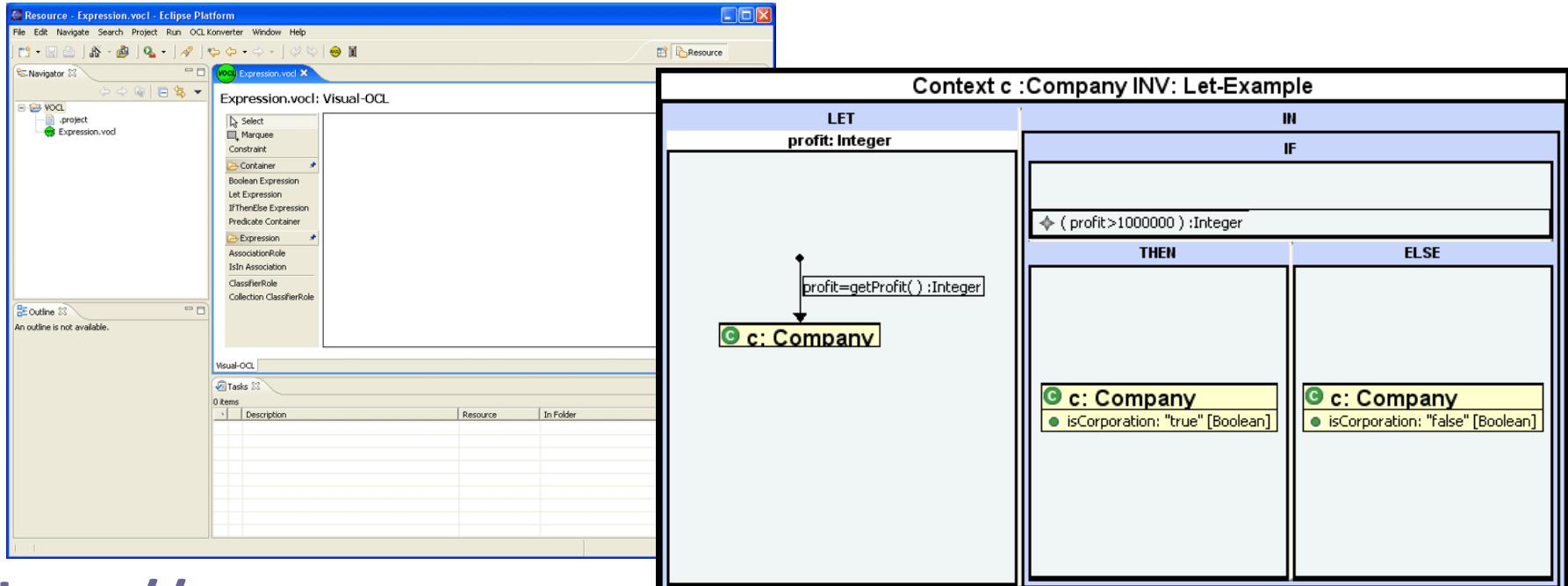


<https://projects.eclipse.org/projects/modeling.mdt.ocl>

Outils OCL

Quelques outils pour la vérification des contraintes OCL :

- **VisualOCL** (A Visual Notation of the OCL) - Université technique de Berlin

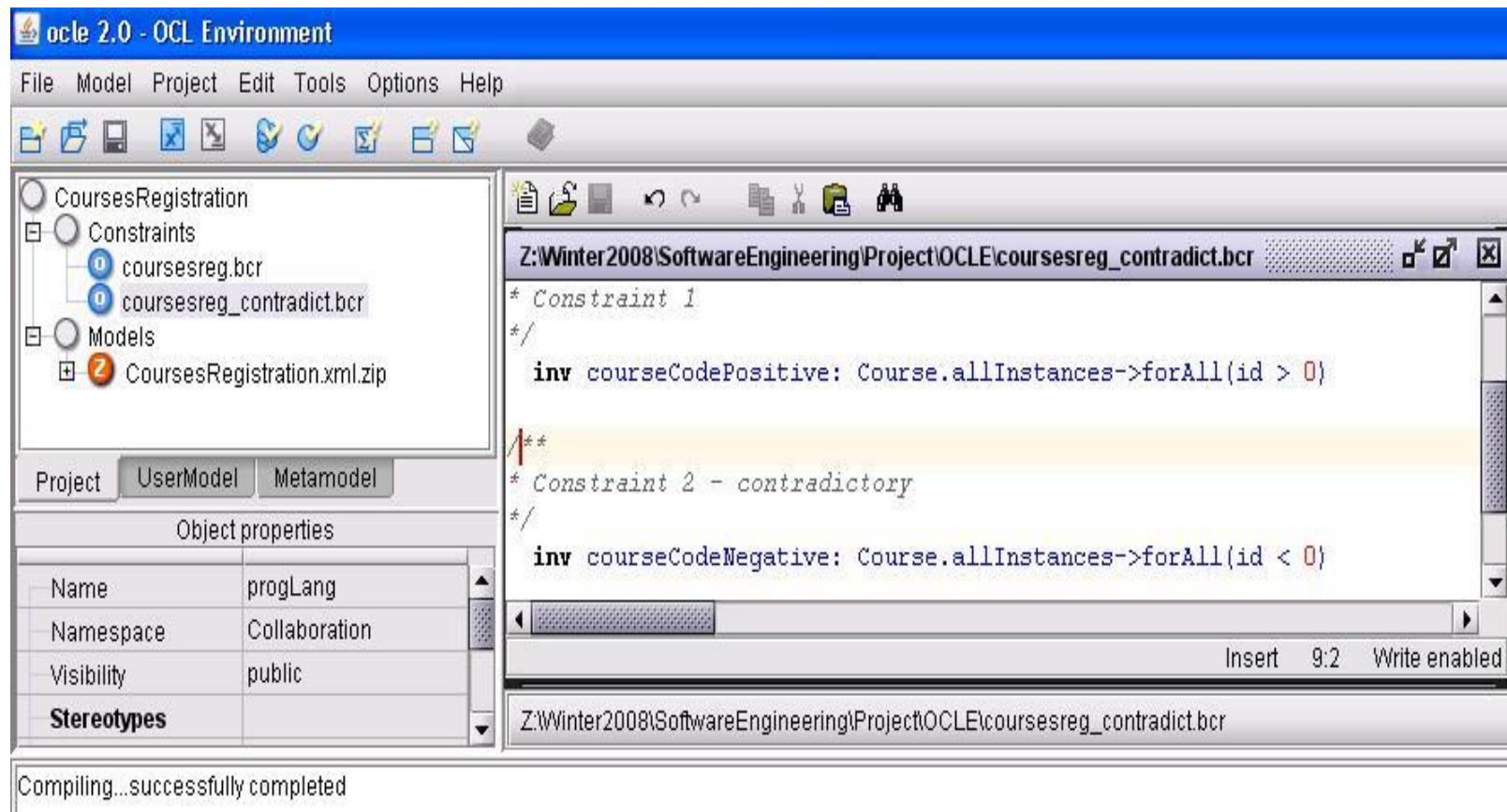


<http://www.user.tu-berlin.de/o.runge/tfs/projekte/vocl/index.html>

Outils OCL

Quelques outils pour la vérification des contraintes OCL :

- **Ocle** (OCL Environment) - Université Babeş-Bolyai (Roumanie)

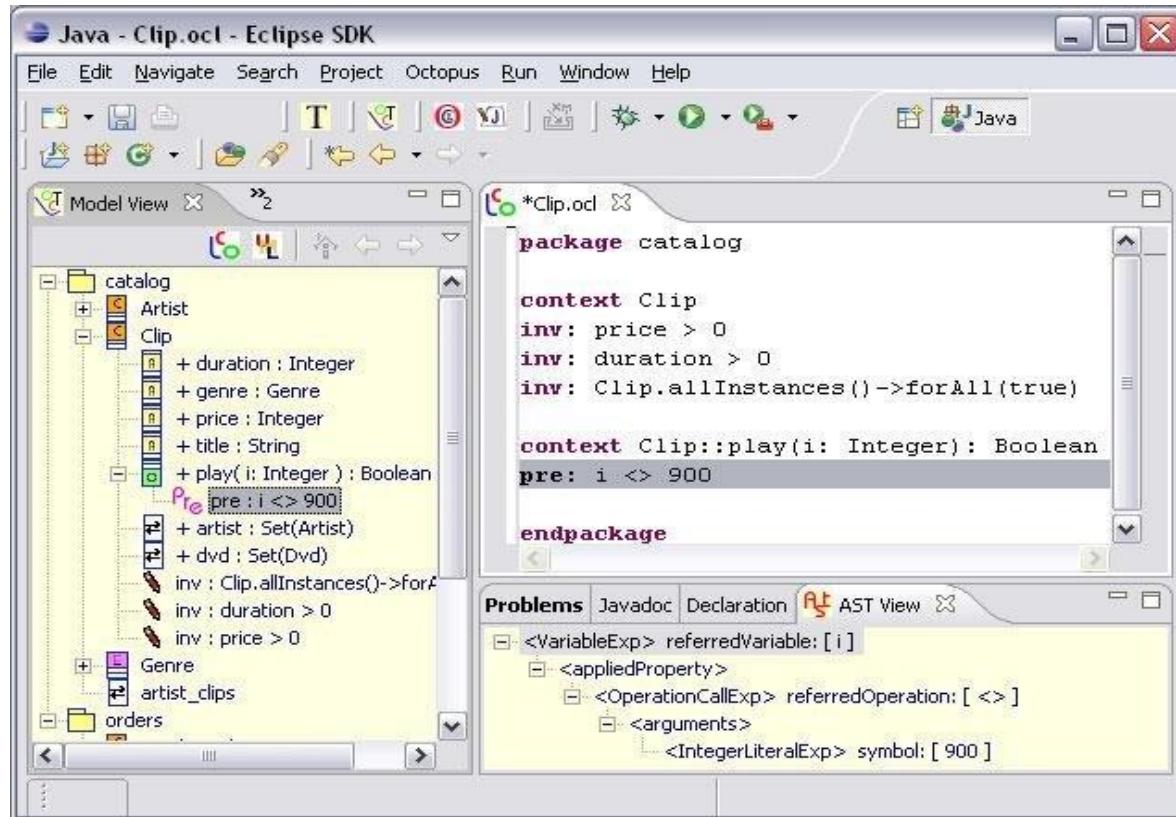


<https://ici.cs.ubbcluj.ro/ocle/>

Outils OCL

Quelques outils pour la vérification des contraintes OCL :

- **Octopus** (OCL Tool for Precise Uml Specifications) - Jos Warmer and Anneke Kleppe

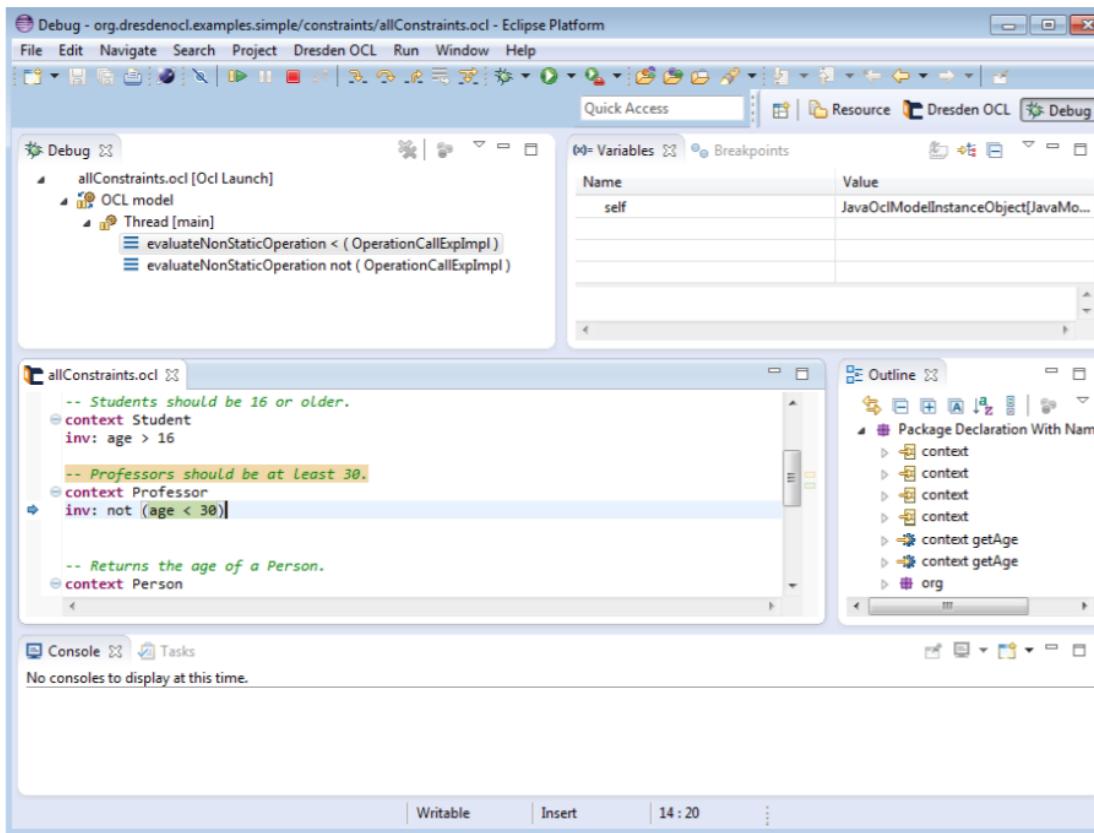


<https://octopus.sourceforge.net/>

Outils OCL

Quelques outils pour la vérification des contraintes OCL :

- **Dresden OCL** : Université technique de Dresde (Allemagne)

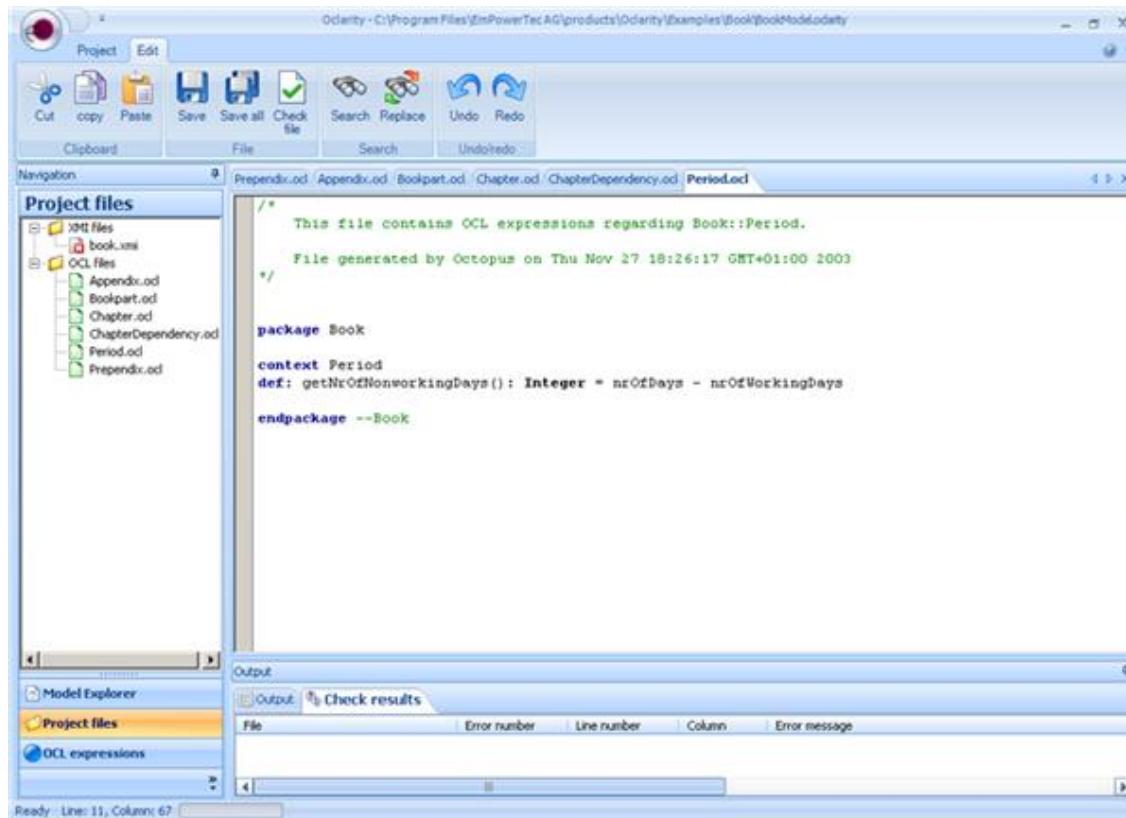


<https://sourceforge.net/projects/dresden-ocl/>

Outils OCL

Quelques outils pour la vérification des contraintes OCL :

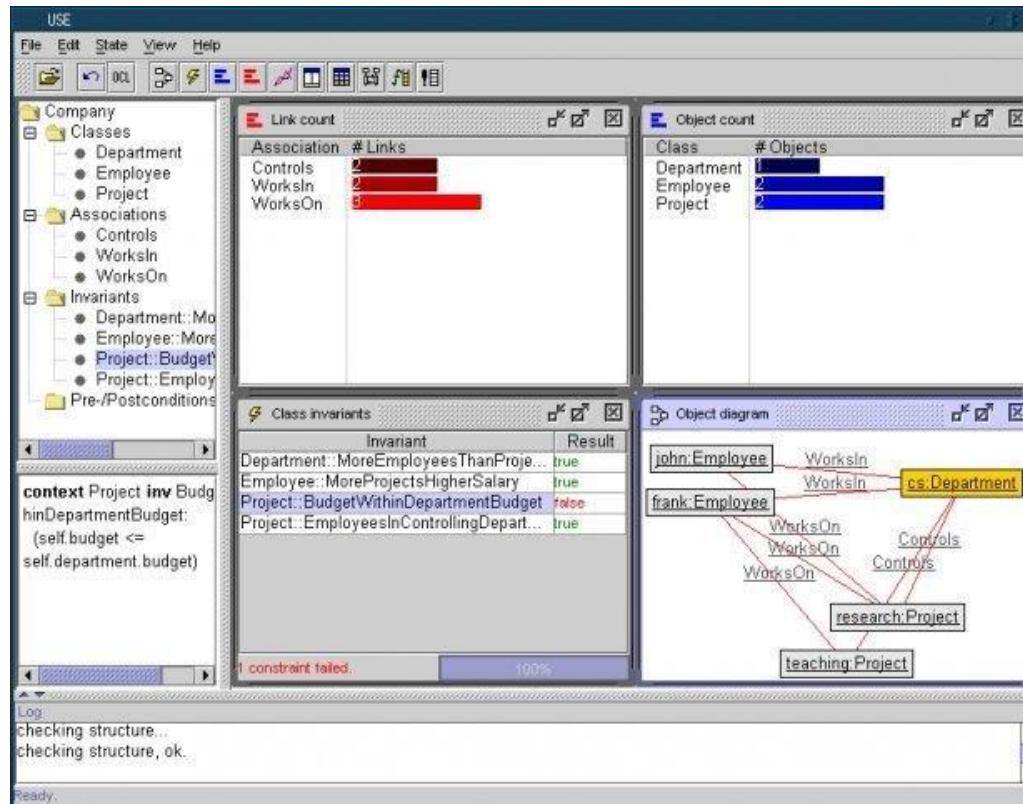
□ **Oclarity** (OCL authoring environment)



Outils OCL

Quelques outils pour la vérification des contraintes OCL :

- ❑ **USE** (UML-based Specification Environment) - Université de Brême (Allemagne)



<https://sourceforge.net/projects/useocl/>

Outils OCL

Quelques outils spécifiques qui intègrent OCL :

- **HOL-OCL** (An Interactive Proof Environment for OCL) -
Université de Sheffield (Angleterre)
 - Intégration de contraintes OCL dans un outil de vérification HOL (Higher Order Logic)
 - <https://www.brucker.ch/projects/hol-ocl/>

The screenshot shows the HOL-OCL interface. At the top is a menu bar with File, Edit, Options, Buffers, Tools, Preview, LaTeX, Command, X-Symbol, and Help. Below the menu is a toolbar with icons for State, Context, Goal, Retract, Undo, Next, Use, Goto, Q.E.D., Find, Command, Stop, Restart, Info, and Help. The main window contains a LaTeX editor with the following code:

```
\begin{small}
\listinput[style=ocl]{company.ocl}
\end{small}

\begin{figure}
\centering
\includegraphics[scale=.6]{company}
\caption{A company Class Diagram\label{fig:company_classdiag}}
\end{figure}

load_xmi "company_ocl.xmi"

thm Company:Person.inv.inv_19_def

lemma "F Company:Person.inv self → Company:Person.inv.inv_19 self"
apply(simp add: Company:Person.inv_def
      Company:Person.inv.inv_19_def)
apply(auto)
*** company.thy 80% (45,14) SWN-27978 (Isar script [PDFLaTeX/F] MMM XS:holocl/s Scripting)----6:35 2.39
\sync:thm Company:Person.inv.inv_19_def; \sync;
Person.inv.inv_19 =
λself. ∀ p2 ∈ OclAllInstances
    self • ( ∀ p1 ∈ OclAllInstances
        self • ((p1 `>` p2) →
            (Company:Person.lastName p1 `>>` Company:Person.lastName p2)))
```

Below the editor is a terminal window showing the command-line interface:

```
*response* ATI (6,101) (response)----6:35 2.39 Mail -
```

Outils OCL

Quelques outils spécifiques qui intègrent OCL :

- **mOdCL** (Maude + OCL)
 - F. Duran and M. Roldan. *The mOdCL Evaluator*. 2008
 - Intégration des contraintes OCL dans Maude
 - Maude : outil de spécification formelle sur la logique de réécriture
 - <http://maude.lcc.uma.es/mOdCL/>
- **Nomos Software OCL Business Rules for XML**
 - Description de règles OCL sur des données XML
 - <http://nomos-software.com/>
- ...

Outils OCL

Quelques outils qui n'existent plus !!!

- ❑ **EOS** (Eye OCL Software)- A Java component for OCL evaluation
 - ❑ M. Clavel, M. Egea and M. A. G. de Dios. *Building an Efficient Component for OCL Evaluation*. ECEASST 15. 2008
- ❑ **RocLET**
 - ❑ C. Jeanneret, L. Eyer, S. Marković, T. Baar. *RocLET – A Tool for Wrestling with OCL Specifications*. In: Proc. 9th Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS'06). 2006
- ❑ **Incremental OCL**
 - ❑ T. Vajk, G. Mezei, T. Levendovszky. *An incremental OCL compiler for modeling environments*. In: Electronic communications of the EASST. OCL Concepts and Tools 2008
- ❑ **kent ocl library**
 - ❑ D. Akehurst, O. Patrascoiu. *OCL 2.0 – Implementing the Standard for Multiple Metamodels*. Electronic Notes in Theoretical Computer Science, 2004
- ❑ **MIP OCL2 Parser** (MIP MDA Tools), **OSLO** (Open Source Library for OCL), **SimpleOCL tool**, ...

Conclusion

- OCL est un standard OMG
- C'est un langage formel de description des contraintes sur des modèles orientés objet
- Éviter les expressions OCL trop compliquées
 - Éviter les navigations complexes
 - Bien choisir le contexte (associer l'invariant au bon type!)
- **Dernière version :** OCL 2.4 - Février 2014

<http://www.omg.org/spec/OCL/2.4>